Assignment 2
Queuing System Simulation

Team Members :
Ajinkya (213050034), Arnav (213059002)

Outline

1. System Assumptions
2. Constants , user Inputs and Enums
3. Classes
4. Basic Logic (FCFS)
5. Basic Logic (Round Robin)

System Assumptions

System Type - Closed system

Each User Issues a Fixed number of Requests

Number of cores - 4

Number of Maximum Threads per core - 4

Request Buffer Size - 100

Constants User inputs and Enums

Constants-

- Max_buffer_Size
- Max_thread_count
- Conext_switch_time
- Max_Request_Generated

Users Inputs –

- Mean_interarrival,
- Mean_service
- Number_of_users

Scheduling Policy *(Enum)*

- FCFS (1)
- Round Robin (2)

Server Status *(Enum)*

- Idle (1)
- Busy (2)

Event Types *(Enum)*

- Arrival (1)
- Departure (2)
- Context_Switch_In (3)

Distribution Type *(Enum)*

- Exponential (1)
- Uniform (2)
- Constant (3)

Classes

Service_time

- *Attributes:*
  - typeOfDistribution(Enum Distribution type)
- *Methods:*
  - getServiceTime() /*Generation function{describes the distribution}*/

Timeout

- *Attributes:*
  - constantTime (double)
  - typeOfDistribution(Enum Distribution type)
- *Methods:*
  - getTimeoutTime() /*Generation function{describes the distribution}*/

Classes

Event

- Attributes:
  - arrival_time (double)
  - timeout(double)
  - serviceTime
  - core (int)
  - thread(int)
  - response_count
- Methods:
  - getRandomThinkTime()/*random value chosen in range [4,10]*/
  - getRemainingServiceTime()

Classes

Core

- *Attributes:*
  - threads [Max Thread Count] (Event Object List)
  - status (int) {Server Status}
  - thread_busy_count (int)
- *Methods*:
  - GetCoreStatus()
  - setCoreStatus()
  - addToThread()
  - removeFromThread()
  - getBusyThreadCount()
  - setBusyThreadCount()

Classes

Scheduler

- *Attributes:*
  - Type (int) {Scheduling policy}
  - Context_switch_time (double)
- *Methods:*
  - switching the threads ()

Server

- *Attributes:*
  - Core Object [4];
  - service_time Object;
  - Scheduler Object;
  - {Waiting Buffer} Event Obj queue [Max buffer size] (shared among all cores)
- *Methods:*
  - getNextEventFromBuffer()
  - getServerStatus()
  - setServerStatus()
  - getCoreObj()

Classes

Event Handler

- *Attributes*:
  - Server Obj
  - timing_next_event[Max_event_count] (a priority queue of tuples <event_time, event obj> prioritized on event_time)
  - Timeout Obj
- *Methods*:
  - getNextEvent()
  - manageEvent()
  - Arrive()
  - Depart()
  - getServerObj()
  - setEvent()

Base Logic (FCFS)

```
Main Function ()
{
    read_input() // take user input
    Initialize() //Initialize the simulation

    Event_handler_Obj = new Event_Handler();

    While (No requests < max Request count){
        Event_handler_obj.Next event();
        Event_handler_obj.manageEvent();
    }
}
```

Base Logic (FCFS)

```
Function Arrival(Event X)
{
    if(no of user < max user){
        Create new EVENT obj and assign arrival time
    }
    if(server busy){
        Put Event X in Event queue;
    }
    Else{
        Set the departure time of the event X.
    }
}
```

Base Logic (FCFS)

```
Function Departure (Event X)
{
     if(Event queue empty){
          Set status idle  of the core belonging to the
          event X that called up the departure
     }
     Else{
          Remove event A from the waiting buffer
          Set the departure time of the event A.
     }
     If Event X -> departure time <  Event X -> timeout{
          Event X -> Response_count++;
     }
     /*Get the Event Object Ready of next Request*/
     Set think-time of event X  randomly
     Set the arrival time of event X based on the think-time
}
```

Base Logic (Round Robin)

```
Main Function ()
{
    read_input() // take user input
    Initialize() //Initalize the simulation

    Event_handler_Obj = new Event_Handler();

    While (No requests < max Request count){
        Event_handler_obj.Next event();
        Event_handler_obj.manageEvent();
    }
}
```

Base Logic (Round Robin)

```
Function Arrival(Event X)
{
    if(no of user < max user){
        Create new EVENT obj and assign arrival time
    }
    if(server busy){
        Put Event X in Event queue;
    }
    Else{
        If Scheduler.timeQuantum > event X.remainingServiceTime()
            Set contextinTime for Event X.
        else
            Set departureTime for Event X.
    }
}
```

Base Logic (Round Robin)

```
Function Departure (Event X)
{
     if(Event queue empty){
          Set status idle  of the core belonging to the
          event X that called up the departure
     }
     Else{
          Remove event A from the waiting buffer
          If Scheduler.timeQuantum > event A.remainingServiceTime()
               Set contextinTime for Event A.
          else
               Set departureTime for Event A.
     }
     If Event X -> departure time <  Event X -> timeout{
          Event X -> Response_count++;
     }
     /*Get the Event Object Ready of next Request*/
     Set think-time of event X  randomly
     Set the arrival time of event X based on the think-time
}
```

Code Highlights

Server Log Output

Trace.txt

```
88.3154    [A A A I ]  EMPTY     Cntx Swtch In          88.4154
==========================================================================
88.4154    [A A A I ]  EMPTY     Departure              88.4806
==========================================================================
88.4806    [A I A I ]  EMPTY     Arrival                88.7719
==========================================================================
88.7719    [A A A I ]  EMPTY     Departure              88.787
==========================================================================
88.787     [I A A I ]  EMPTY     Cntx Swtch In          88.8719
==========================================================================
88.8719    [I A A I ]  EMPTY     Cntx Swtch Out         89.3719
==========================================================================
89.3719    [I A A I ]  EMPTY     Cntx Swtch In          89.4719
==========================================================================
89.4719    [I A A I ]  EMPTY     Departure              89.7606
==========================================================================
89.7606    [I A I I ]  EMPTY     Cntx Swtch Out         89.9719
==========================================================================
89.9719    [I A I I ]  EMPTY     Cntx Swtch In          90.0719
==========================================================================
90.0719    [I A I I ]  EMPTY     Departure              90.953
==========================================================================
90.953     [I I I I ]  EMPTY     Arrival                91.2877
==========================================================================
91.2877    [A I I I ]  EMPTY     Cntx Swtch In          91.3877
==========================================================================
91.3877    [A I I I ]  EMPTY     Arrival                91.4158
==========================================================================
91.4158    [A A I I ]  EMPTY     Cntx Swtch In          91.5158
==========================================================================
```

EventHandler’s ManageEvent

```cpp
void EventHandler::manageEvent(Event event){

    switch (event.type)
    {
     case ARRIVAL:
         this->arrive(event);
         break;

     case DEPARTURE:
         this->depart(event);
         break;

     case CONTEXTSWITCHIN:
         contextSwitchIn(event);
         break;

     case CONTEXTSWITCHOUT:
         contextSwitchOut(event);
         break;

     default:
         break;
    }
}
```

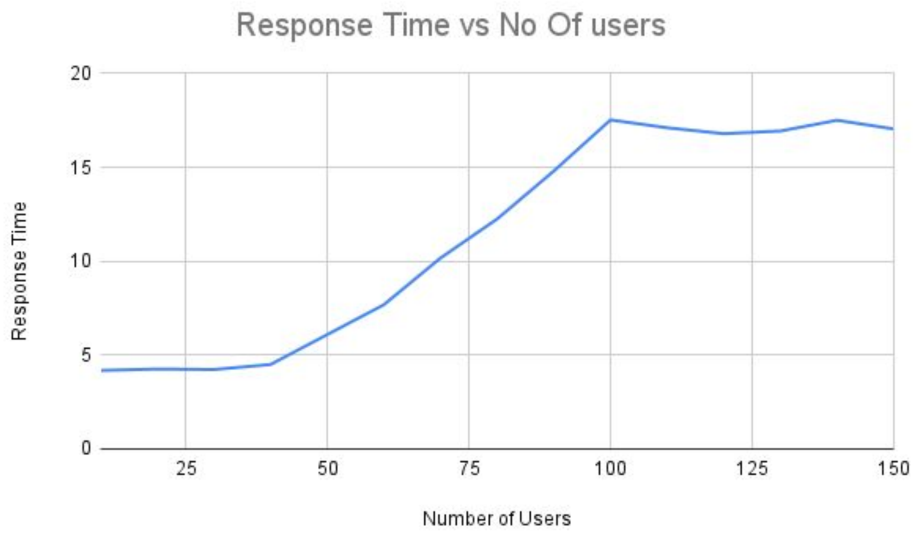
Results

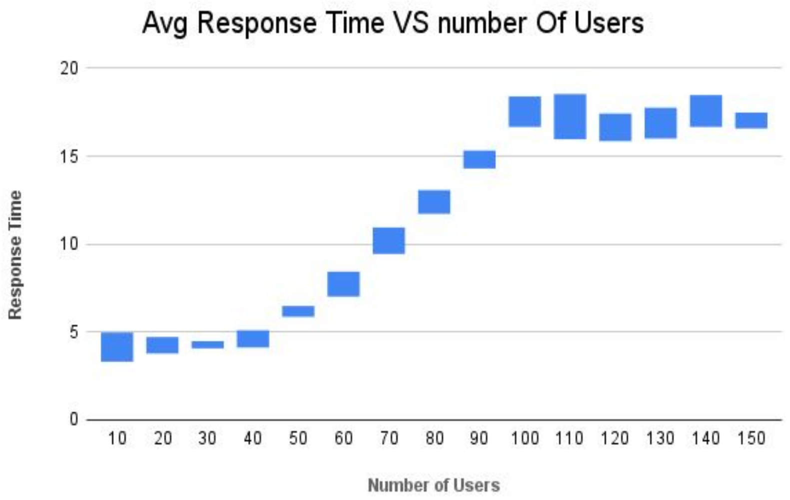
- What will be the Average Response Time of the described system?

➢ To find out the average response time, we ran the simulation for 5 times with randomly chosen Service Times, Timeouts and Think Time for particular number of users. In every run we calculated the mean response time and finally taken the average of these 5 means.

➢ The same experiment was repeated for variable number of users and the average of 5 runs for every user is plotted as confidence interval in the graph shown.

Response Time Vs Users





Line Plot Shows Response time getting saturated after MAX user reaches 100.

Confidence Plot shows increase of variation in response time as the Number of User increases.

Results

- What will be the Throughput of the described system?

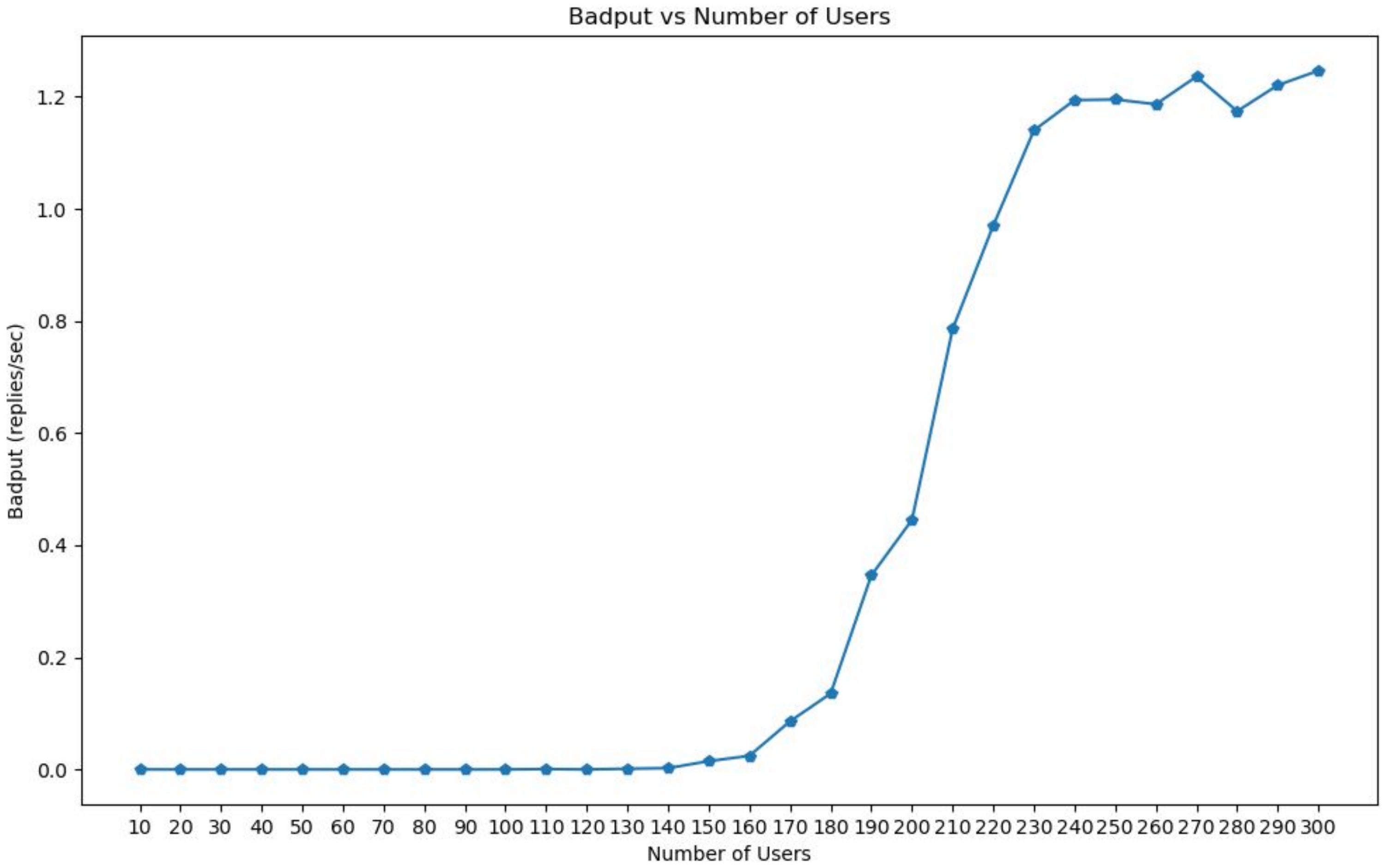
➢ To calculate find throughput, we first of all counted the number of requests timed out and then completed in retries and number of requests completed without any retries. From these two values, we calculated

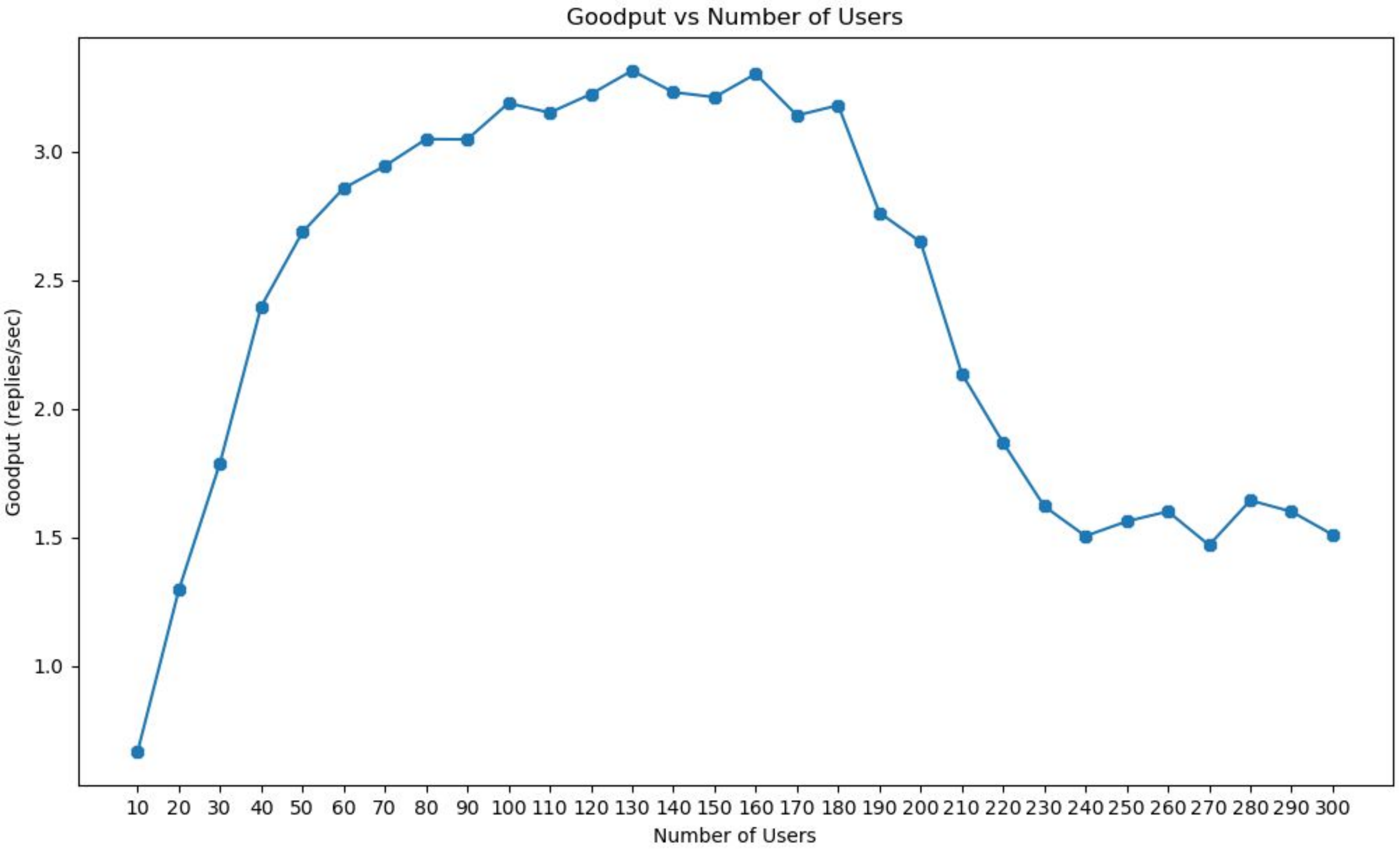
Badput = number of requests timed out/total simulation time

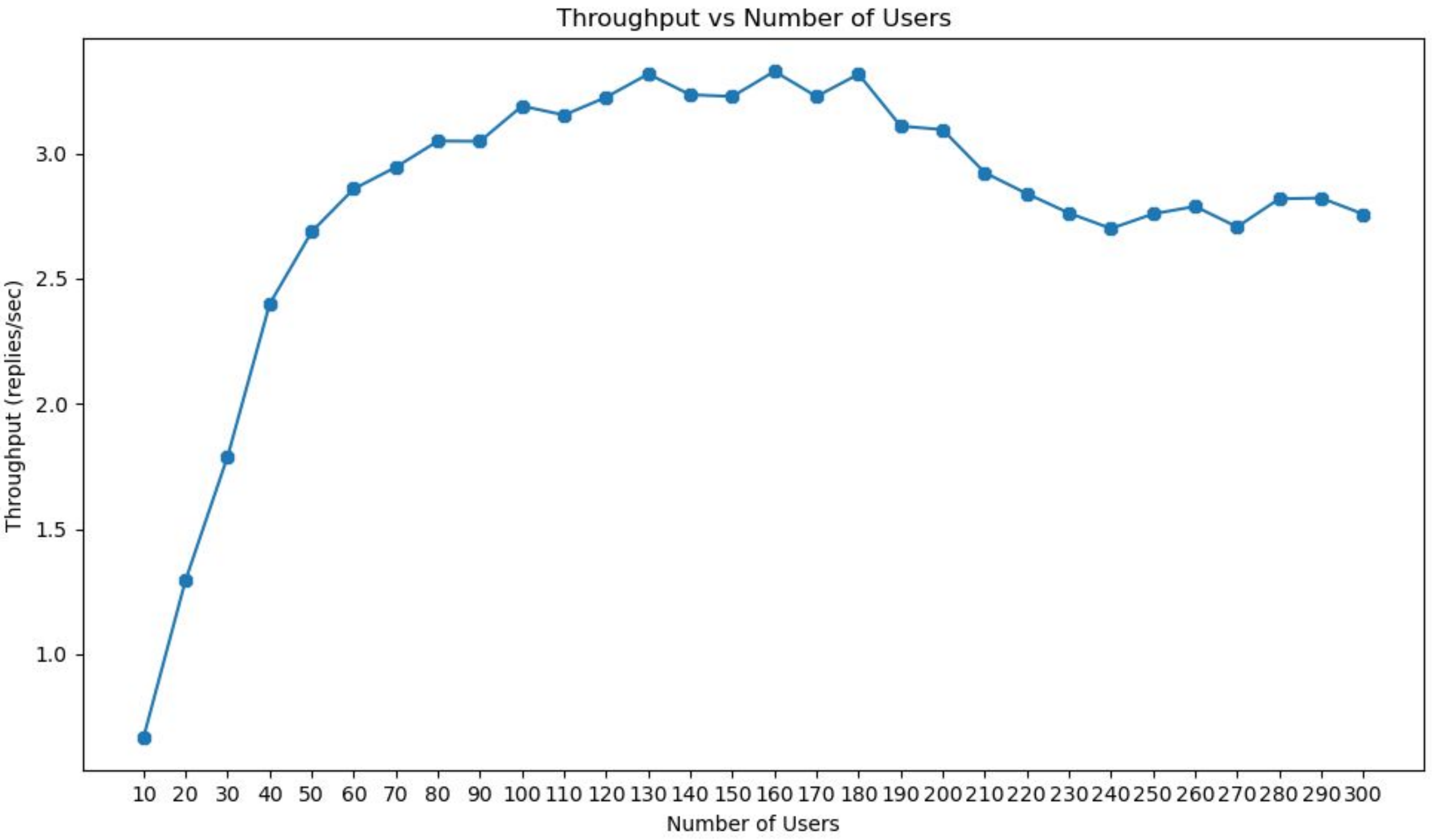
Goodput = number of requests without timeouts/total simulation time

Throughput = Goodput + Badput

The corresponding graphs are shown below


Badput vs Number of Users
Badput (replies/sec)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300
Number of Users


Goodput vs Number of Users
Goodput (replies/sec)
Number of Users
1.0
1.5
2.0
2.5
3.0
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300

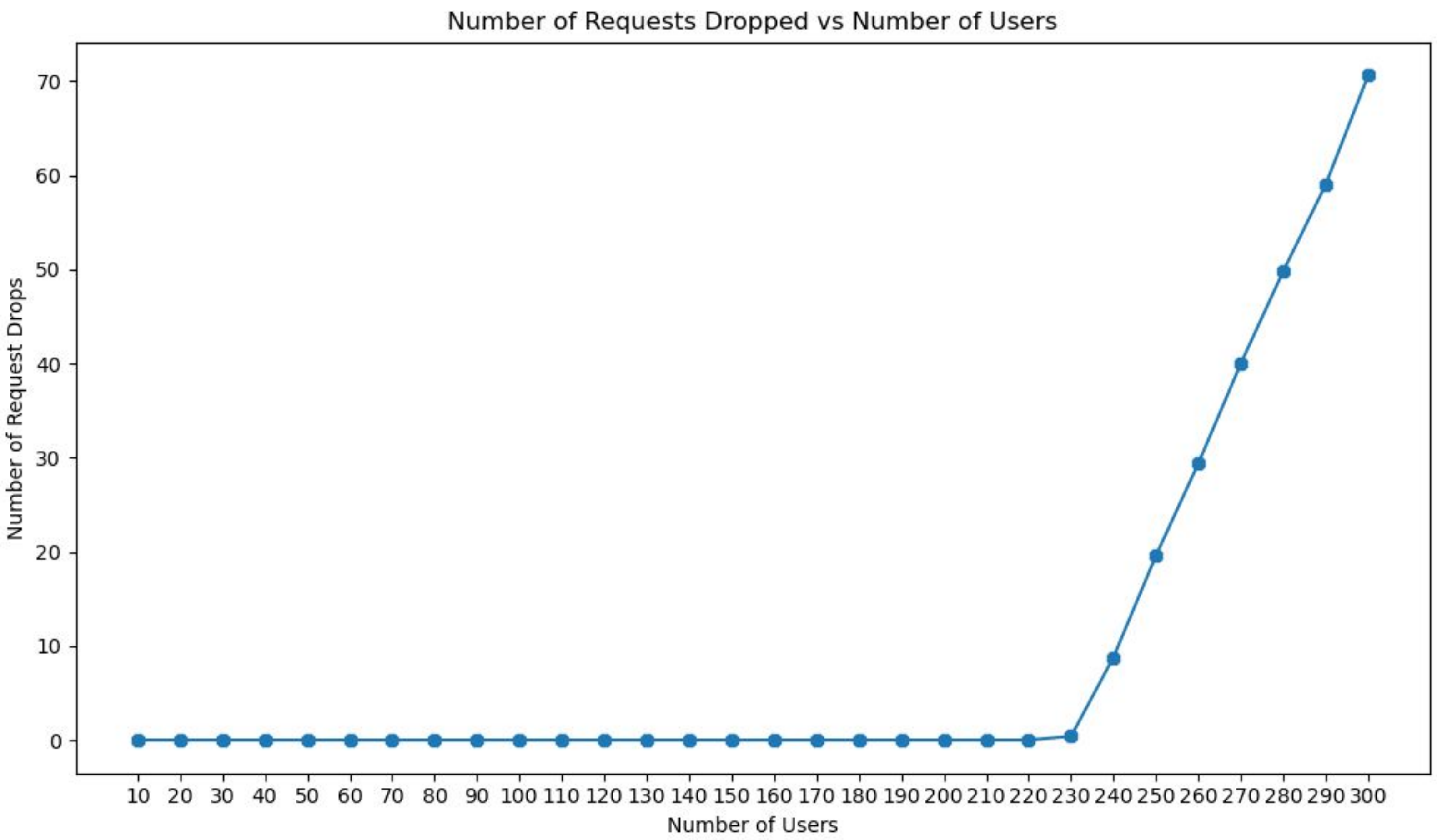


Throughput vs Number of Users
Throughput (replies/sec)
1.0
1.5
2.0
2.5
3.0
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300
Number of Users

Results

- What will be the Average Number of Request Drops of the described system?

➢ We counted the number of requests which got dropped due to lack of space in request buffer and plotted the graph of number of requests dropped against number of users. The graph is shown below.



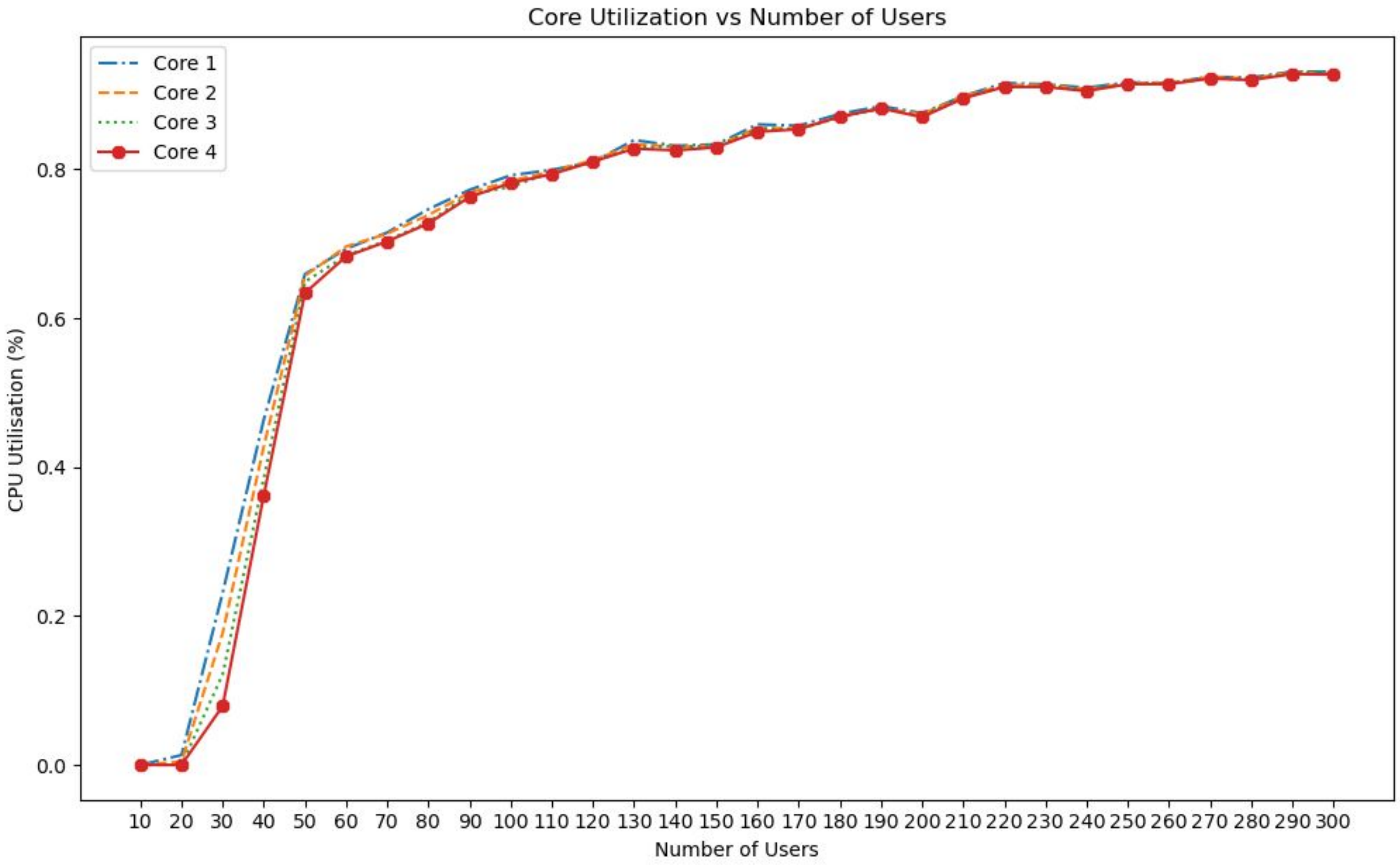
Number of Requests Dropped vs Number of Users
Number of Request Drops
0
10
20
30
40
50
60
70
Number of Users
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300

Results

- What will be the Average Core Utilisation of the described system?
- ➢ For finding core utilization, we calculated the number of threads active at every time step on each core and divided it by max number of threads on that core. This gave us the utilization in that time span. To find Average utilisation for each core we used following formula

Avg Core Utilization = difference between events * (Number of threads / Max number of threads)

We calculated utilisation for each core and plotted it against number of users.


Core Utilization vs Number of Users
Core 1
Core 2
Core 3
Core 4
CPU Utilisation (%)
0.0
0.2
0.4
0.6
0.8
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250 260 270 280 290 300
Number of Users

Conclusion

We successfully implemented simulation program for closed queuing system.

We ran some experiments to check and verify following metrics

- Average Response Time
- Throughput
- Average Request Dropped
- Average Core Utilization

The experimental values we got are nearer to the theoretical calculations.